**Results**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Run | Results | | | | Data | | | |
| tp | fp | tn | fn |
|
| Precision | Recall | Accuracy | F1 |
| 1 | 0 | 0 | 0.871 | 0 | 0 | 247 | 1707 | 5 |
| 2 | 0 | 0 | 0.444 | 0 | 0 | 1084 | 870 | 5 |
| 3 | 0.00409 | 0.0867 | 0.532 | 0.00780 | 13 | 3169 | 3751 | 137 |
| 4 | 0.00802 | 0.676 | 0.0679 | 0.0159 | 23 | 2845 | 185 | 11 |
| 5 | 0 | 0 | 0.996 | 0 | 0 | 0 | 1421 | 6 |
| 6 | 0 | 0 | 0.986 | 0 | 0 | 0 | 1092 | 16 |
| 7 | 0 | 0 | 0.00451 | 0 | 0 | 1087 | 5 | 16 |

**Details**

**Run 1: MSUpdaterTrojanWhitepaper.conll**

Train size 47, dev size 21

With 5 manually annotated articles,

Accuracy might have been low because of a lack of normalization of the text entities (eg PittyTiger is automatically annotated but not Pitty\_Tiger)

**Run 2: MSUpdaterTrojanWhitepaper.conll**

Same train size, same dev size

Changed main label in config.conf to B-Malware

**Run 3: apt28.conll**

Accuracy might be low because of the large number of noise within the data, a result of parsing from tika.

Script written after run 3 to remove all non alpha-numeric characters. Minimum word frequency was also lowered to account for entities that appear very very rarely.

Config.conf file as shown below

[config]

dataset = fcepublic

path\_train = train

#Maybe.tsv

path\_dev = dev

#2q-report-on-targeted-attack-campaigns.tsv

path\_test = test

#Agent.BTZ\_to\_ComRAT\_fake.tsv

conll\_eval = True

main\_label = B-Malware

model\_selector = dev\_conll\_f:high

preload\_vectors = 'GoogleNews-vectors-negative300.txt'

word\_embedding\_size = 100

crf\_on\_top = False

emb\_initial\_zero = False

train\_embeddings = True

char\_embedding\_size = 100

word\_recurrent\_size = 300

char\_recurrent\_size = 100

hidden\_layer\_size = 50

char\_hidden\_layer\_size = 50

lowercase = True

replace\_digits = True

min\_word\_freq = 3

singletons\_prob = 0.1

allowed\_word\_length = -1

max\_train\_sent\_length = -1

vocab\_include\_devtest = True

vocab\_only\_embedded = Falsex

initializer = glorot

opt\_strategy = adadelta

learningrate = 0.01

clip = 0.0

batch\_equal\_size = False

max\_batch\_size = 32

epochs = 200

stop\_if\_no\_improvement\_for\_epochs = 7

learningrate\_decay = 0.9

dropout\_input = 0.5

**Run 4: MiniDuke\_Paper\_Final.conll**

Same settings as run 3

**Run 5: Democracy\_HongKong\_Under\_Attack.conll**

Changed mainlabel to 0, changed model\_selector to dev\_f.

**Run 6: Sandworm\_briefing2.conll**

Same settings. added more files to use as dataset for both.

Total of 56 for train and 42 for dev.

**Run 7: Sandworm\_briefing2.conll**

Changed main label to B-Malware

Same data set as above.

**Reasons for the failed model:/Other Observations**

Too little data

Some words didn't have embeddings. Can try training with a higher number of text, with a GPU to speed up epochs. Results shown in the paper were achieved after training the model with 10000 abstracts, while in comparison, a maximum of 100 were used this time.Thus, names of malware etc appear too few times in the corpus eg 4-5 detected times a passage. Most words will be unlabeled. Recommendation to use more data.

Variation dependent on main label

When the model runs with O as main label, the number of true negatives will be high but this results in a comparatively lower number of entities being identified. However, when B-Attacker/Malware is used as the main label, the number of false positives increases greatly. This may be due to a relatively small number of labelled entities labelled automatically via script. Recommendation is to annotate more articles by hand in order to feed the model with more accurate data.

Lack of Hyperparameter Tuning

Hyperparameters such as layer size were entirely untouched. Results might be better with more tweaking of the hyperparameters.

Embeddings

Word embedding dataset used was enwiki\_20180420\_100d.txt, while experiments in the paper were ran with Google’s 300d embedding dataset. Given the smaller size of the dataset, some malware might have lacked pre-loaded embeddings. Indeed, while running the code,

**n\_preloaded\_embeddings: 11724**

**parameter\_count: 3850404**

**parameter\_count\_without\_word\_embeddings: 2092404**

Is shown.

Evaluation standards

With the parameter ‘model\_selector’ in config.conf, the model shows learning when ‘dev\_f:high’ is selected rather than ‘dev\_f\_conll:high’ as recommended by the paper. In the latter, the best epoch was always epoch 0.

**How to run**

WebAnn

* Download WebAnno 3.6.4 standalone version
* Open in browser with the ip address given
* Default username and password is both ‘admin’
* Go to projects, and create a project with project type ‘annotation’
* Go under tagsets, and add the relevant tags ‘malware, attacker, tool’ or any others as needed
* Go under documents and upload documents. Be sure to amend the format accordingly. List of formats supported can be seen from the drop-down menu.
* Go back home, and click annotation to start annotating.
* Upload document to start annotating.
* Highlight a word to go to the annotation menu. From there, select the appropriate label.
* Once done, export the file.
* \*\*For all tests, conll2002 format is used as export format.

Script

* Put input files into the train and dev folders. Conll, text or tsv files are fine, as long as they adhere to the format specified in the readme.
* In line 292 of labeller.py, I directly specified the embedding file to be run, ignoring the config settings directly. This was due to a path-related issue I had when I first tried running the script.
* Ensure that both sets of data have the same sets of labels, eg if the label ‘I-Attacker’ is present in the train corpus, it should appear in the dev corpus as well, or else an error related to ‘UNK token’ will occur.
* With VSCode, enter the seq labelling folder and run python experiment.py config.conf
* After the model is trained, the model will be saved inside the folder with the name you choose in config.conf, under ‘save’.
* To run a test model, run the command python print\_output.py labels model\_file input\_file. The input file should be inside the sequence labelling folder. The output will be a text file called ‘result.txt’, where the rightmost column is the prediction given by the model.
* A readme is included in the folder as well for more details on the config.conf file.